ATLAS
MANUAL DE USUARIO
Pruebas con Selenium

Versión 1.2

Arquitectura de Software

Agencia de Informática y Comunicaciones de la Comunidad de Madrid

Hoja de Control

Título	Manual de Pruebas con Selenium		
Documento de Referencia	NORMATIVA ATLAS		
Responsable	Área de Aplicaciones Especiales y Arquitectura de Software		
Versión	1.2	**Fecha Versión**	30/04/2014

Registro de Cambios

Versión	Causa del Cambio	Responsable del Cambio	Fecha del Cambio
1.0	Versión inicial del documento	Arquitectura de Aplicaciones	12/06/2013
1.1	Eliminado comentario que sobraba	Arquitectura de Aplicaciones	18/11/2013
1.2	Añadido 5.1 Test unitarios para aplicaciones con certificados	Arquitectura de Aplicaciones	11/01/2014

Índice

1. INTRODUCCIÓN 4

1.1. AUDIENCIA OBJETIVO 4
1.2. CONOCIMIENTOS PREVIOS 4

2. DESCRIPCIÓN 5

3. EJECUCION DE LAS PRUEBAS 6

3.1. EJECUCIÓN CONTRA EL ENTORNO LOCAL 6
3.2. EJECUCIÓN CONTRA ALGÚN ENTORNO DE LOS DE ICM 10

4. CONFIGURACIÓN 14

4.1. CONFIGURACIÓN DEL NAVEGADOR DE PRUEBAS 14
4.1.1. Configuración para el navegador Internet Explorer 15
4.1.2. Configuración para el navegador Google Chrome 16

5. CREACION DE TESTS UNITARIOS 17

5.1. OBTENCIÓN DE PROPIEDADES DE ENTORNO 17
5.2. AUTENTICAR AL USUARIO EN LA APLICACIÓN 19
5.3. CREACIÓN DE UN TEST UNITARIO DE FORMA MANUAL 19
5.3.1. Operaciones comunes 23
5.4. CREACIÓN DE UN TEST UNITARIO CON SELENIUMIDE 28
5.4.1. Ejemplo de test 30
5.4.2. Ejecución de un test grabado 31
5.4.3. Exportación del test a una clase java 32
5.4.4. Adaptar test grabado 33
5.1. TEST UNITARIOS PARA APLICACIONES CON CERTIFICADOS 36
5.1.1. Crear un profile de Firefox para las pruebas 36
5.1.2. Importar el certificado a usar 36
5.1.3. Hacer los test de Selenium con el nuevo perfil por defecto 37

6. ENLACES RELACIONADOS 38

1. INTRODUCCIÓN

El framework Atlas de la Comunidad de Madrid proporciona herramientas para el testeo del software a varios niveles. En el nivel de pruebas unitarias, se deben implementar los tests con junit para asegurar el correcto funcionamiento de los servicios de la aplicación. En el nivel de funcionalidad sobre navegador, la herramienta a utilizar es **Selenium, que permite reproducir de manera automática el comportamiento que tendría un usuario final utilizando la aplicación desde un navegador.**

En este documento se describirán los procedimientos para la creación de los tests funcionales con Selenium y la ejecución de éstos sobre los navegadores soportados por la herramienta.

La creación y ejecución de estos tests sólo tiene sentido en aplicaciones web, por tanto se realizarán sobre los proyectos basados en arquetipos web del framework Atlas. Un arquetipo web de Atlas tiene dos módulos:

- **Módul web**: para la creación de la aplicación
- **Módulo test**: para el desarrollo de los tests funcionales con Selenium, lo que se trata en este documento.

1.1. Audiencia objetivo

Este documento está orientado a desarrolladores java que participen en la creación de una aplicación web con el framework Atlas.

1.2. Conocimientos Previos

Para un completo entendimiento del documento, el lector deberá tener conocimientos previos sobre las siguientes tecnologías:

- Spring Framework.
- Creación de tests unitarios.
- Conocimiento de la herramienta Selenium, a través de su documentación

2. DESCRIPCIÓN

En este documento se describirá la forma de realizar los tests de Selenium de dos formas diferentes:

- A través de una **programación tradicional** donde se localizarán los elementos a comprobar y donde se realizarán validaciones que decidirán si los tests son correctos o no.
- Con el **uso de la herramienta Selenium IDE** para generar los mismos tests y la forma de traducirlos a código java que pueda ejecutarse dentro del proyecto de forma automatizada con maven.

Todos los tests realizados con Selenium serán dependientes del entorno y se parametrizarán en el fichero environment.properties aquellos datos que sean diferentes en cada ejecución, según se describe más adelante en este documento.

3. EJECUCION DE LAS PRUEBAS

Los arquetipos *web* y *documentumweb* vienen ya configurados (en su sub-módulo *test*) para la ejecución de pruebas con Selenium. La configuración inicial de los arquetipos ejecuta las pruebas de Selenium **sólo desde el módulo test (nunca desde el módulo raíz del arquetipo web)**. Estas pruebas están configuradas por defecto para ser ejecutadas sobre el navegador *Firefox*. Más adelante en este documento se describirá como ejecutar las pruebas sobre los navegadores *Google Chrome* e *Internet Explorer*.

AVISO IMPORTANTE
Todos los pasos descritos en este manual se refieren al sub-módulo “test” del arquetipo web, nunca se realizan sobre el submódulo “web”.

Los tests del arquetipo vienen configurados por defecto para usar el navegador *Mozilla Firefox*. Si se quiere utilizar este navegador no es necesario realizar ningún cambio de configuración.

Una vez creado el arquetipo, se pueden ejecutar los tests de forma rápida realizando los siguientes pasos, dependiendo de si queremos ejecutar los tests contra nuestro entorno local o contra uno de los entornos de ICM.

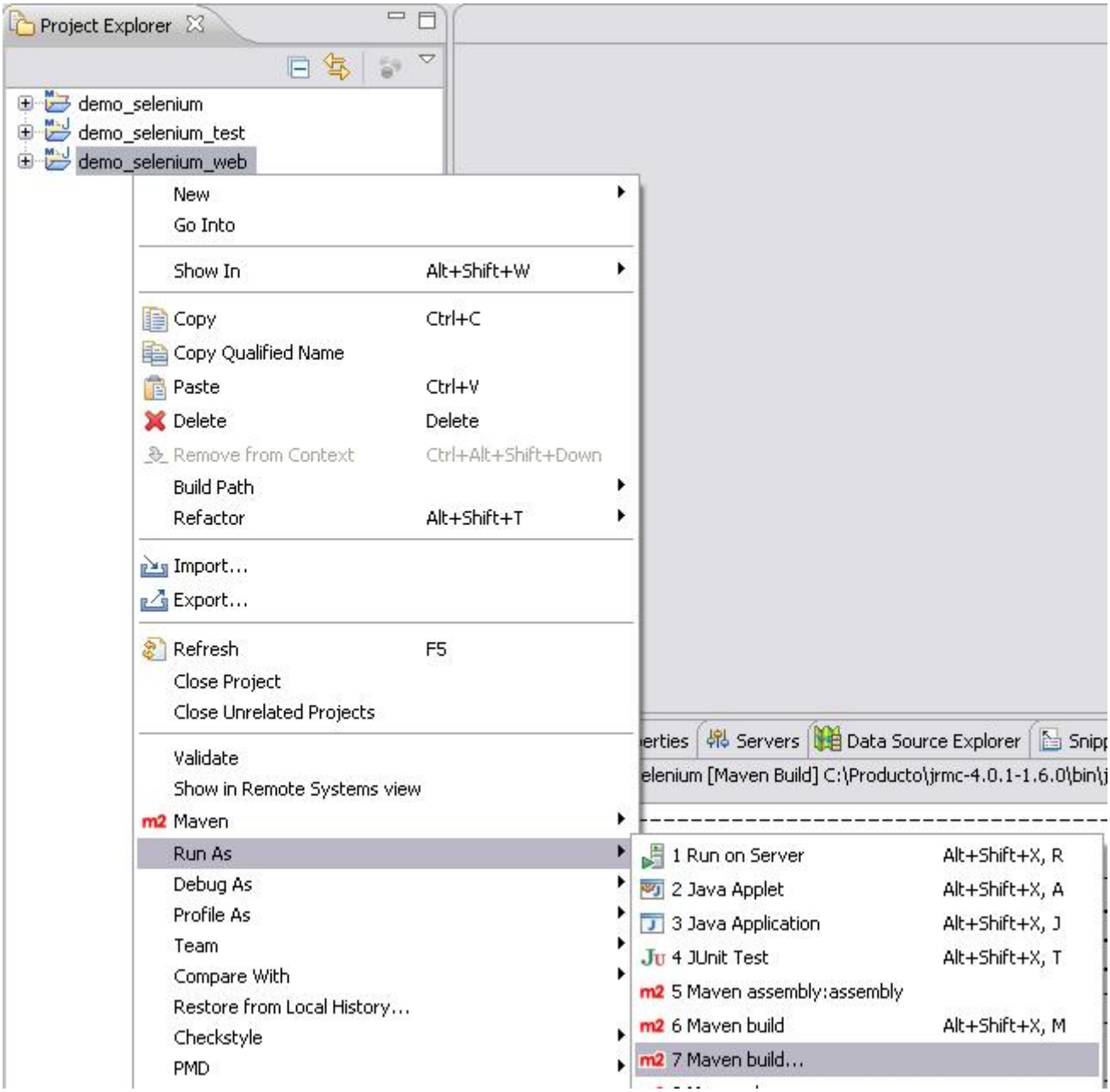
3.1. Ejecución contra el entorno local

Seguir este apartado para ejecutar los tests contra nuestro entorno local. Si queremos ejecutarlos contra uno de los entornos de ICM, proceder con el apartado 3.2.

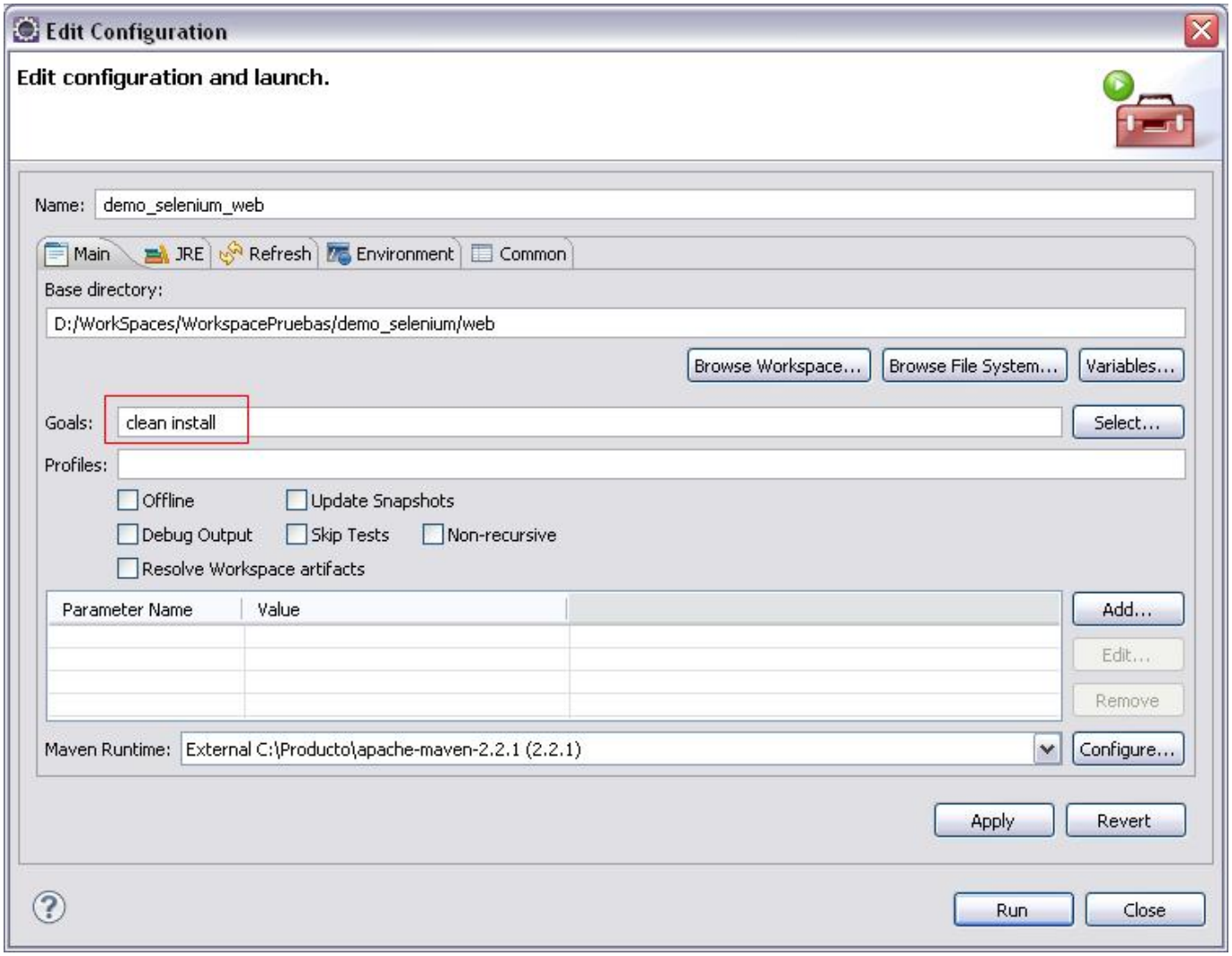
Paso 1: construir la aplicación web desde el subproyecto “web” del proyecto

Antes de ejecutar los tests en el *entorno local* es necesario generar la aplicación, construyendo el proyecto a través de maven.

En Eclipse, pulsamos con el botón derecho sobre el **proyecto web** y elegir la opción de menú "*Run -> Run As -> Maven build…*".

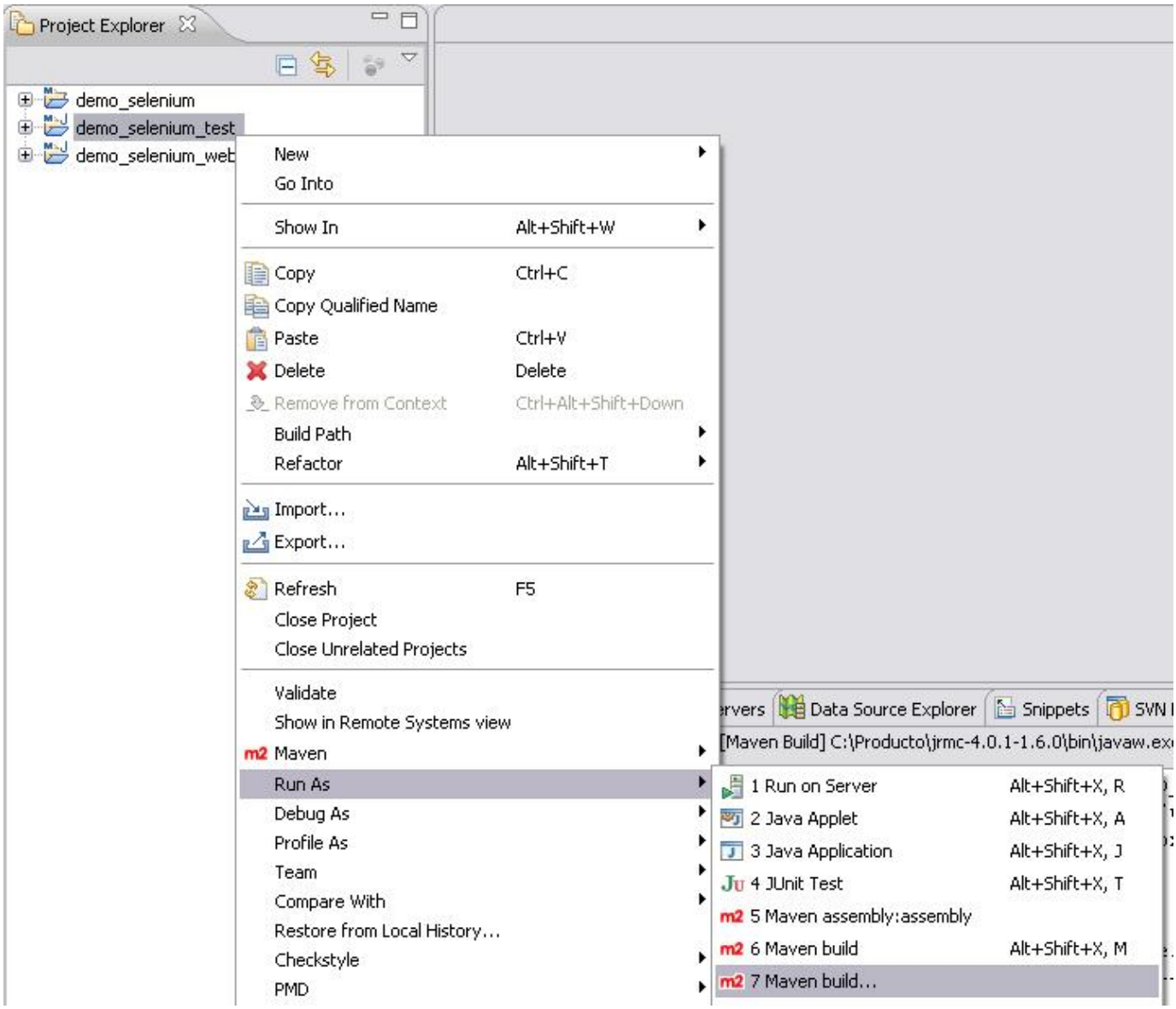


En la pantalla, indicar "*clean install*" en la zona "Goals", y pulsar sobre "*Run*" según se muestra en esta figura:

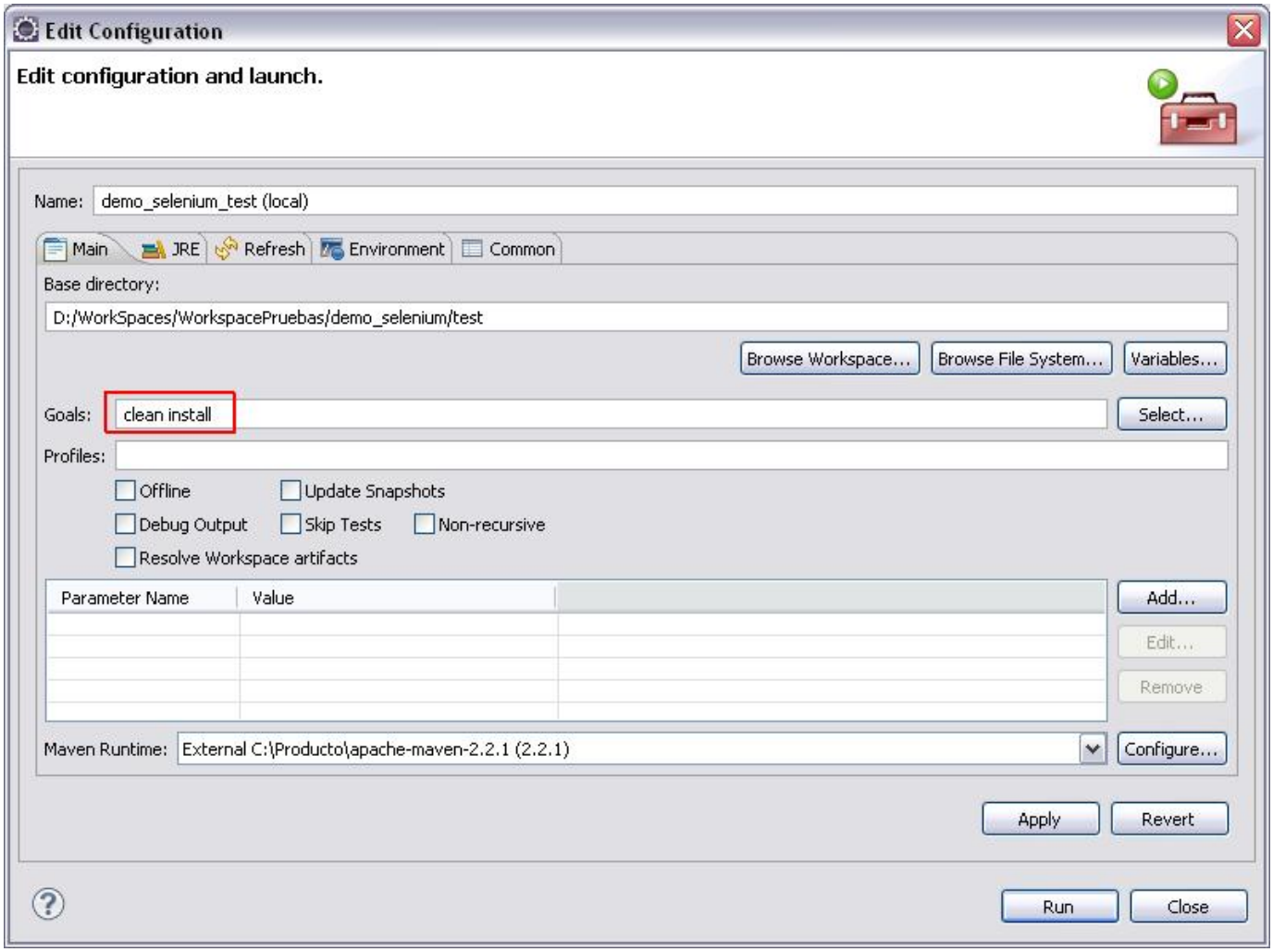


Paso 2: ejecutar los tests de Selenium en el módulo test

Una vez generada la aplicación en el entorno local, debemos de ejecutar los tests de Selenium. Para ello, pulsamos con el botón derecho sobre el **sub-proyecto test** y elegir la opción de menú "*Run -> Run As -> Maven build…*".



En la pantalla, indicar "*clean install*" en la zona "Goals". Pulsar sobre "*Run*" según se muestra en esta figura:



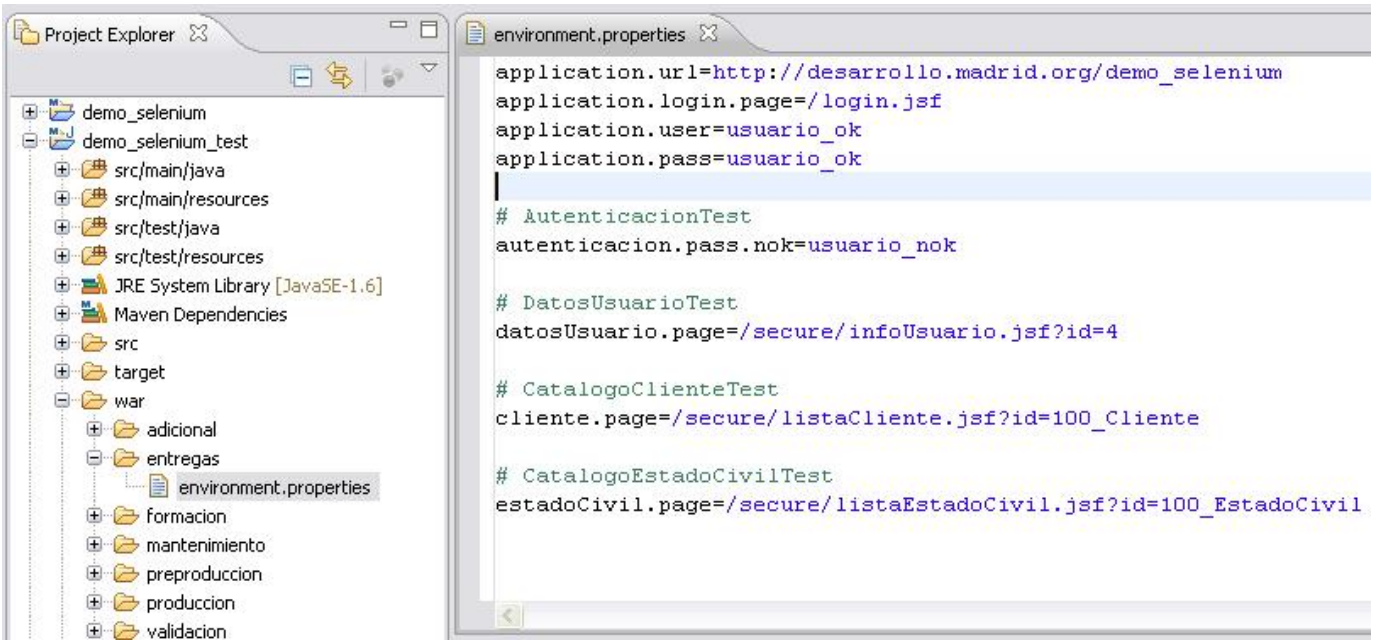
Este comando desencadenará la ejecución de los tests de Selenium en el entorno local. Podrá verse como el proceso de prueba abre y cierra navegadores ejecutando los tests configurados.

3.2. Ejecución contra algún entorno de los de ICM

Seguir este apartado para ejecutar los tests contra alguno de los entornos de ICM. Si queremos ejecutarlos contra el entorno local del desarrollador, proceder con el apartado 3.1.

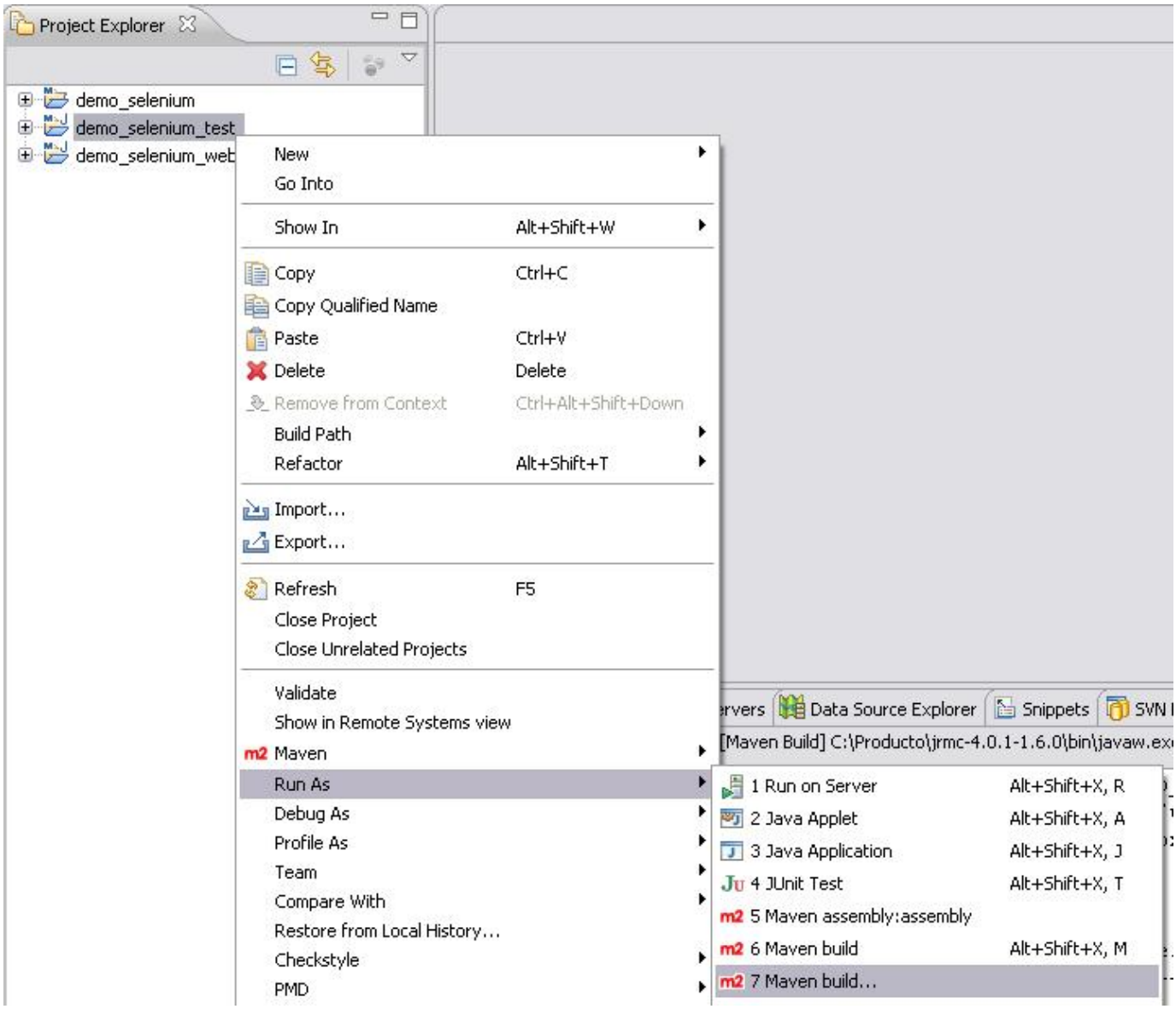
Paso 1: configurar el fichero environment.properties correspondiente al entorno

Antes de ejecutar los tests contra uno de los entornos de ICM, debemos revisar que la configuración del fichero *environment.properties* (dentro del **sub-módulo test**) es correcta. Para ello revisamos el fichero *war/{nombreEntorno}/environment.properties*, donde {nombreEntorno} es el correspondiente al entorno contra el que queremos ejecutar los tests de Selenium:



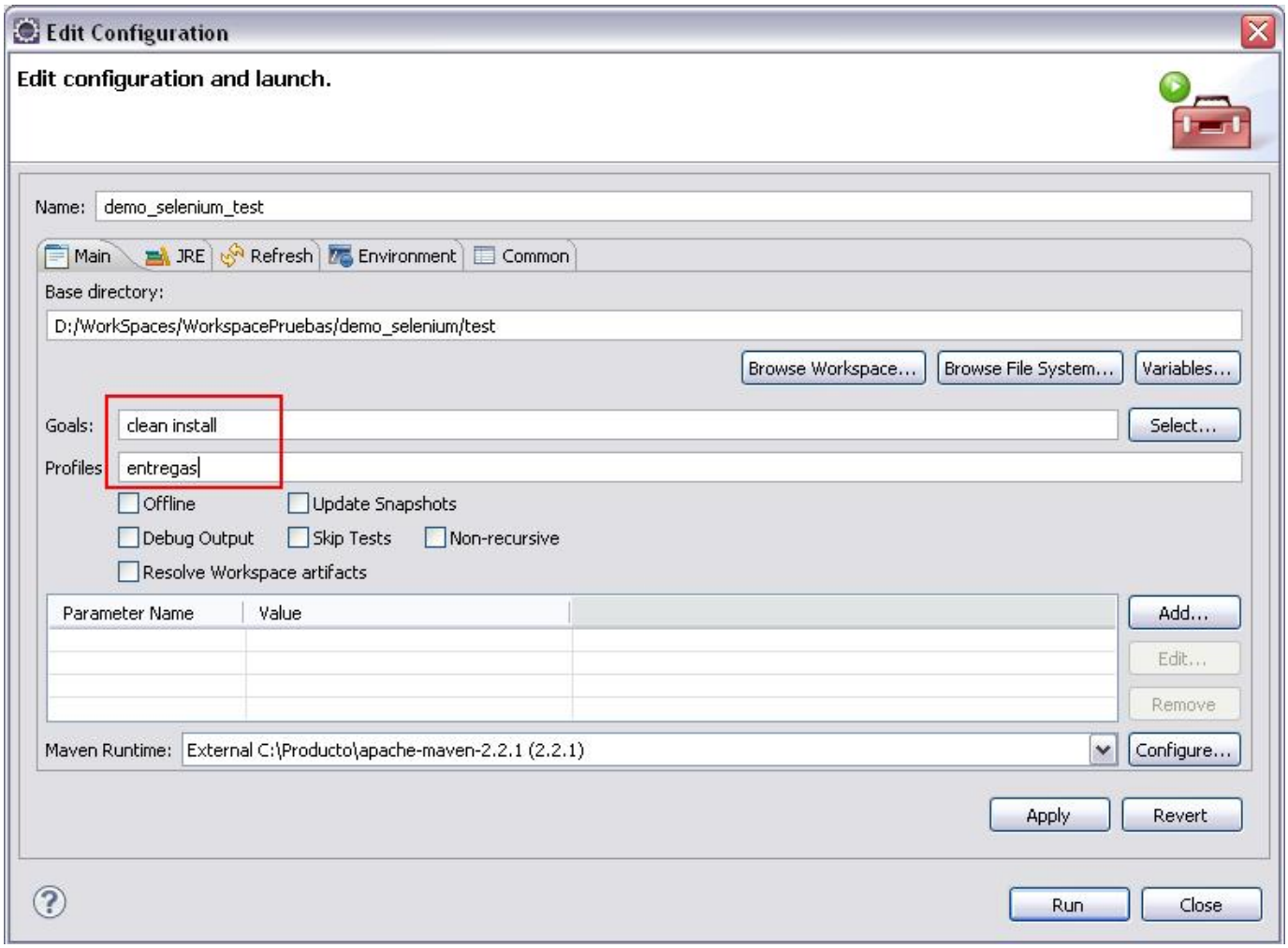
Paso 2: ejecutar los tests de Selenium en el módulo test

Una configurados los tests, debemos de ejecutar los tests de Selenium. Para ello, pulsamos con el botón derecho sobre el **sub-proyecto test** y elegir la opción de menú "*Run -> Run As -> Maven build*…".



En la pantalla, indicar "*clean install*" en la zona "Goals". Además rellenamos la casilla '*Profiles'* con el perfil del entorno que se quiera probar (ej. '*entregas'* para probar en entorno de desarrollo, o '*validacion'* para el entorno de validación).

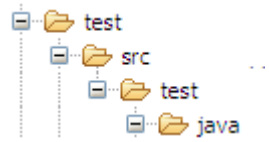
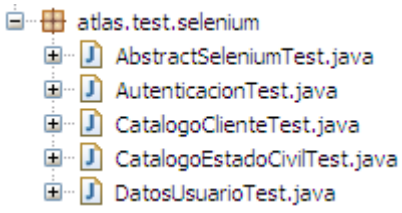
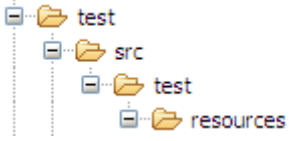
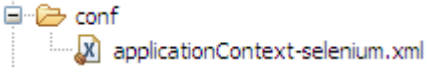
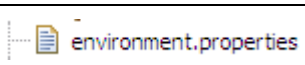
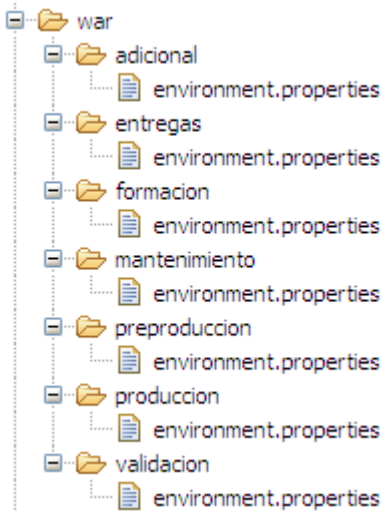
Pulsar sobre "*Run*" según se muestra en esta figura:



Este comando desencadenará la ejecución de los tests de Selenium. Podrá verse como el proceso de prueba abre y cierra navegadores ejecutando los tests configurados.

4. CONFIGURACIÓN

El módulo *test* de un arquetipo web o documentumweb contiene los siguientes elementos para la ejecución de las pruebas de Selenium:

<table>
<tr><td rowspan="2"> 
</td><td colspan="2">Ruta del módulo de pruebas de Selenium donde residen las clases de prueba</td></tr>
<tr><td> 
</td><td>Clases de prueba de Selenium. Hay una clase base de tests llamada AbstractSeleniumTest.java</td></tr>
<tr><td rowspan="3"> 
</td><td colspan="2">Ruta del módulo de pruebas de Selenium donde residen los ficheros de configuración.</td></tr>
<tr><td> 
</td><td>Configuración de Spring donde se establecerá el navegador de pruebas.</td></tr>
<tr><td> 
</td><td>Fichero de configuración con los datos de prueba del entorno local.</td></tr>
<tr><td> 
</td><td colspan="2">Ficheros de configuración con los datos de pruebas para el resto de entornos.</td></tr>
</table>

4.1. Configuración del navegador de pruebas

Los tests del arquetipo recién generado vienen configurados por defecto para usar el navegador *Mozilla Firefox*. Si se quiere utilizar este navegador no es necesario realizar ningún cambio de configuración.

Si se quiere utilizar otro navegador, la configuración se realiza en el fichero *src/test/resources/conf/applicationContext-selenium.xml*. Este es el contenido del fichero:

src/test/resources/conf/applicationContext-selenium.xml

```
<?xml version="1.0" encoding="UTF-8"?>


<beans ...>

    [...]

  <!-- Navegador para las pruebas de Selenium -->
  <bean id="driver" class="org.openqa.selenium.firefox.FirefoxDriver" ... />

  <!-- ATENCION: configuración para IExplorer o Google Crome -->
  <!--
  <bean id="driver" class="org.openqa.selenium.ie.InternetExplorerDriver" ... />
  <bean id="driver" class="org.openqa.selenium.chrome.ChromeDriver" ... />
  -->

</beans>
```

Para más información sobre el uso de distintos navegadores puede consultarse la documentación oficial de Selenium. Puede consultarse también la tabla de compatibilidad del Selenium con los distintos navegadores y Sistemas Operativos.

4.1.1. Configuración para el navegador Internet Explorer

La realización de los tests sobre el navegador Internet Explorer se realiza a través de un ejecutable .exe que tiene que estar instalada en una ruta en el path de sistema.
En esta dirección (https://code.google.com/p/selenium/wiki/InternetExplorerDriver) está la documentación de uso de esta librería. A continuación se describen los pasos a realizar:

4.1.1.1. Descargar la librería de interfaz

En el momento de creación de este documento, la Url de descarga de la librería de interfaz está en la dirección:

https://code.google.com/p/selenium/downloads/list

El paquete que ha de descargarse es IEDriverServer_Win32_2.32.3.zip. La versión de la librería (en este caso la 2.32.3) puede cambiar.

4.1.1.2. Descomprimir el paquete zip de la librería

Crear la siguiente carpeta: *C:/Producto/Selenium* y descomprimir el fichero zip en este directorio. El paquete zip solo contiene un fichero llamado *IEDriverServer.exe*.

4.1.1.3. Configurar la carpeta Selenium en el path de sistema

Para que el ejecutable pueda ser localizado sin problemas la ruta *C:/Producto/Selenium* debe ser añadida en la

variable *PATH* de Windows.

4.1.1.4. Establecer Internet Explorer como navegador de pruebas

Para ello habrá que editar el fichero *applicationContext-selenium.xml* y cambiar la definición del bean '***driver'***.

src/test/resources/conf/applicationContext-selenium.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>

    [...]

  <!-- Navegador para las pruebas de Selenium -->
  <bean id="driver" class="org.openqa.selenium.ie.InternetExplorerDriver" ... />

</beans>
```

Una vez realizado esto, ya se pueden ejecutar los tests de Selenium y el navegador a utilizar será Internet Explorer.

4.1.2. Configuración para el navegador Google Chrome

La realización de los tests sobre el navegador Google Chrome se realiza a través de un ejecutable .exe que tiene que estar instalada en una ruta en el path de sistema. A continuación se describen los pasos a realizar:

4.1.2.1. Descargar la librería de interfaz

En el momento de creación de este documento, la Url de descarga de la librería de interfaz está en la dirección:

https://code.google.com/p/chromedriver/downloads/list

El paquete que ha de descargarse es chromedriver_win_26.0.1383.0.zip. La versión de la librería (en este caso la 26.0.1383.0) puede cambiar.

4.1.2.2. Descomprimir el paquete zip de la librería

Crear la siguiente carpeta: *C:/Producto/Selenium* y descomprimir el fichero zip en este directorio. El paquete zip solo contiene un fichero llamado *chromedriver.exe*.

4.1.2.3. Configurar la carpeta Selenium en el path de sistema

Para que el ejecutable pueda ser localizado sin problemas la ruta *C:/Producto/Selenium* debe ser añadida en la variable *PATH* de Windows.

4.1.2.4. Establecer Google Chrome como navegador de pruebas

Para ello habrá que editar el fichero *applicationContext-selenium.xml* y cambiar la definición del bean '***driver'***.

src/test/resources/conf/applicationContext-selenium.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>

    [...]

  <!-- Navegador para las pruebas de Selenium -->
  <bean id="driver" class="org.openqa.selenium.chrome.ChromeDriver" ... />

</beans>
```

Una vez realizado esto, ya se pueden ejecutar los tests de Selenium y el navegador a utilizar será Google Chrome.

5. CREACION DE TESTS UNITARIOS

Los arquetipos web y documentumweb contienen unas clases de test de ejemplo y diversas utilidades para ayudar a crearlos. Estos son:

- *AbstractSeleniumTest.java*: clase base para la creación de tests de Selenium. Esta clase proporciona facilidades para la apertura del navegador y el posicionamiento en la página de pruebas, además de realizar el login en la aplicación de forma semiautomática. Incluye además métodos de utilidad para localizar componentes JSF por su identificador. **Todos los tests de Selenium tendrán que extender esta clase.**
- *AutenticacionTest.java*: Test de ejemplo que realiza un login correcto y otro incorrecto en la aplicación para testear el acceso.
- *DatosUsuarioTest.java*: Test de ejemplo que navega por la página de datos del usuario para comprobar su funcionalidad.
- *CatalogoClienteTest.java*: Test de ejemplo que comprueba las funcionalidades de la página de gestión de cliente.
- *CatalogoEstadoCivilTest.java*: Test de ejemplo que comprueba las funcionalidades de la página de gestión de cliente.

5.1. Obtención de propiedades de entorno

Para poder ejecutar los tests en los distintos entornos de ICM es necesario definir un conjunto de datos válido para cada uno de ellos. Por este motivo la gestión de estos datos se realizará a través de un fichero *environment.properties* para cada entorno.

En el módulo test (como en el módulo web), los ficheros *environment.properties* para cada entorno están en la carpeta '*war*' bajo el directorio con el nombre de este.

A diferencia del módulo web, este fichero contendrá todos los datos que quieran parametrizarse en las pruebas. A continuación se muestra el contenido del fichero para el entorno local (situado en src/test/resources):

src/test/resources/environment.properties

```
application.url=http://localhost:9080/
application.login.page=/login.jsf
application.user=usuario_ok
application.pass=usuario_ok

# AutenticacionTest
autenticacion.pass.nok=usuario_nok

# DatosUsuarioTest
datosUsuario.page=/secure/infoUsuario.jsf?id=4

# CatalogoClienteTest
cliente.page=/secure/listaCliente.jsf?id=100_Cliente

# CatalogoEstadoCivilTest
estadoCivil.page=/secure/listaEstadoCivil.jsf?id=100_EstadoCivil
```

Nota: Para el resto de entornos el fichero environment.properties se encuentra en la ruta *war/{nombreEntorno}/environment.properties*, donde {nombreEntorno} es el correspondiente al entorno contra el que queremos ejecutar los tests de Selenium:

En el ejemplo pueden distinguirse dos conjuntos de datos diferenciados:

- Datos con prefijo '*application*': son datos de carácter general que pueden ser usados por todos los tests de Selenium.
- Resto de datos: los datos particulares de cada test están agrupados , con un comentario con el nombre del test en que se usan y un prefijo acorde para evitar conflictos de datos entre distintos tests.

Para hacer disponibles estos datos de forma sencilla en los tests se puede utilizar la anotación *@AutowiredProperty* dentro de las clases Java que contienen los tests.

La anotación puede utilizarse de dos formas:

- Cuando se define sobre un atributo de la clase de test sin indicar nada más, se buscará una propiedad general (esto es, con el prefijo '*application*') con el nombre del atributo. En el siguiente ejemplo, se buscará la propiedad '*application.user*'.

src/test/java/../DatosUsuarioTest.java

```
@ContextConfiguration(locations={"classpath:/conf/applicationContext-
selenium.xml"})
public class DatosUsuarioTest extends AbstractSeleniumTest {

    /** Usuario para el login. Obtenido de environment.properties */
    @AutowiredProperty
    private String user;

[...]
```

- Cuando se define utilizando el atributo "*name*", se especifica el nombre completo de la propiedad a buscar (debe ser siempre así para los datos específicos de cada test) que es '*datosUsuario.page'*.

src/test/java/../DatosUsuarioTest.java

```
@ContextConfiguration(locations={"classpath:/conf/applicationContext-
selenium.xml"})
public class DatosUsuarioTest extends AbstractSeleniumTest {

    /** Página de ejecución del test */
    @AutowiredProperty(name="datosUsuario.page")
    private String page;

[...]
```

5.2. Autenticar al usuario en la aplicación

La clase *AbstractSeleniumTest* tiene un método para realizar la autenticación de forma automática proporcionando las credenciales. A continuación se muestra un ejemplo de autenticación al inicio de un test:

Ejemplo de autenticación en test de Selenium

```
@ContextConfiguration(locations = {"classpath:/conf/applicationContext-
selenium.xml"})
public class DatosUsuarioTest extends AbstractSeleniumTest {

    /** Usuario para el login. Obtenido de environment.properties */
    @AutowiredProperty
    private String user;

    /** Contraseña para el login. Obtenido de environment.properties */
    @AutowiredProperty
    private String pass;


    @Test
    public void testAcceso() throws Exception {

        try {
            // Autenticar si es necesario
            authenticate(this.user, this.pass);

[...]
```

5.3. Creación de un test unitario de forma manual

Existen dos formas de crear tests unitarios con *Selenium*:

- La primera es la creación de las clases java de forma manual, añadiendo código para localizar, accionar y comprobar los elementos en pantalla

- La segunda forma es la utilización del plugin *SeleniumIDE* en *Firefox* para grabar el test al estilo de la grabación de una macro para luego capturar el script y formatearlo en modo *Java-WebDriver* (este modo se verá en el siguiente capítulo).

En este apartado se describirá el proceso para la creación de un nuevo test de forma manual, así como la forma de modelar las acciones más comunes a realizar en una página.
Para crear un test de Selenium de forma manual deben realizarse los siguientes pasos:

Paso 1: Crear la clase de test y extender AbstractSeleniumTest

A continuación se muestra un ejemplo de nuevo test de Selenium. Este ejemplo extenderá AbstracSeleniumTest y se configurará para cargar la configuración de tests de Spring.

Ejemplo de test de Selenium

```
@ContextConfiguration(locations = {"classpath:/conf/applicationContext-selenium.xml"})
public class EjemploSeleniumTest extends AbstractSeleniumTest {

  [...]
```

AVISO IMPORTANTE

Las clases java con los Tests de Selenium deberán situarse dentro de paquetes java en la ruta src/test/java del sub-módulo “test” del arquetipo web.

Paso 2: Parametrizar la página de test

Cada clase de test de Selenium debe, en general, comprobar la funcionalidad de una página. Si este no es el caso (se desea hacer un ciclo de navegación por varias páginas), siempre existirá una página de partida. Esta página se parametrizará en el fichero environment.properties y se proporcionará a través del método *getPage()*.

Ejemplo de test de Selenium

```
@ContextConfiguration(locations = {"classpath:/conf/applicationContext-selenium.xml"})
public class EjemploSeleniumTest extends AbstractSeleniumTest {

    /** Pagina de ejecución del test */
    @AutowiredProperty(name="ejemplo.page")
    private String page;

    /**
     * Paso de la página de ejecución del test a la clase padre
     * @return página de test
     */
    @Override
    protected String getPage() {
        return this.page;
    }


 [...]
```

environment.properties

```
[...]

# EjemploSeleniumTest
ejemplo.page=/secure/ejemplo.jsf
```

El método *getPage()* es abstracto y debe devolver la página de inicio del test. La clase *AbstractSeleniumTest* se encarga de abrir la página en el inicio de cada prueba.

Paso 3: Crear un método de test y autenticar al usuario si es necesario

A continuación se crean las variables de las credenciales y se realiza una autenticación al inicio del test. Para ello, se invoca al método "*authenticate*" heredado de la clase padre:

Ejemplo de test de Selenium

```
@ContextConfiguration(locations = {"classpath:/conf/applicationContext-selenium.xml"})
public class EjemploSeleniumTest extends AbstractSeleniumTest {

    /** Usuario para el login. Obtenido de environment.properties */
    @AutowiredProperty
    private String user;

    /** Contraseña para el login. Obtenido de environment.properties */
    @AutowiredProperty
    private String pass;


    @Test
    public void testEjemplo() throws Exception {

        try {
            // Autenticar si es necesario
            authenticate(this.user, this.pass);


 [...]
```

Paso 4: Completar el test

Solo queda completar el test con las sentencias que probarán la funcionalidad. Como ejemplo, a continuación se muestra código que verifica que la página actual es la correcta y valida el contenido de un campo de texto:

Ejemplo de test de Selenium

```
@ContextConfiguration(locations = {"classpath:/conf/applicationContext-selenium.xml"})
public class EjemploSeleniumTest extends AbstractSeleniumTest {

[...]

@Test
public void testEjemplo() throws Exception {

    try {
      // Autenticar si es necesario
      authenticate(this.user, this.pass);

      // Verificar que la página es la correcta
      WebElement nombre = driver.findElement(By.xpath("//span[contains(text(),'infoUsuario')]"));
      assertNotNull("Pagina incorrecta", nombre);

      // Validar el nombre del usuario
      nombre = getJsfElement("input", "outputTextNombre");
      assertNotNull("Nombre de usuario nulo", nombre);
      assertEquals("Nombre de usuario incorrecto", user, nombre.getAttribute("value"));

    } catch (TimeoutException e) {
      fail("Error cargando página home");

    } catch (NoSuchElementException e) {
      fail("Pagina incorrecta");
    }

}

 [...]
```

5.3.1. Operaciones comunes

A continuación se muestra como realizar las operaciones más comunes sobre páginas web. Para ampliar la información de este apartado se recomienda revisar la documentación online en la web de Selenium.

5.3.1.1. Acceder por URL a una página de la aplicación

Los tests que extienden *AbstractSeleniumTest* ya tienen un mecanismo para abrir la página inicial del test. Si se necesita realizar una carga de una página en base a su URL se puede utilizar el método *openPage* de esta clase.

AbstractSeleniumTest

```
    /**
     * Abre la página sobre la que se realizará el test de Selenium
     */
    protected void openPage();

    /**
     * Abre la página sobre la que se realizará el test de Selenium
     * @param timeoutInSeconds tiempo máximo en segurndos que se esperará
     * a la carga de la página
     */
    protected void openPage(int timeoutInSeconds);

    /**
     * Abre en el navegador la página pasada como parámetros
     * @param url Página a abrir en el navegador
     */
    protected void openPage(String url);

    /**
     * Abre en el navegador la página pasada com parámetro
     * @param url Página a abrir en el navegador
     * @param timeoutInSeconds tiempo máximo en segurndos que se esperará
     * a la carga de la página
     */
    protected void openPage(String url, int timeoutInSeconds);
```

Los métodos que no contienen el parámetro '*url*' abrirán la página proporcionada por *getPage()*. La url que se pasa a este método debe ser absoluta. Para calcular correctamente la *url* si esta es relativa se puede utilizar el método *getPageUrl*.

AbstractSeleniumTest

```
/**
 * Obtiene la URL absoluta de la página pasada.
 * @param page dirección relativa de la página de la que se quiere obtener su URL
 * @return ruta completa a la página
 */
protected String getPageUrl(String page);
```

El *timeout* por defecto para la carga de páginas es de 10 segundos.

5.3.1.2. Localizar un elemento de la página

Cada uno de los tags html de la página web se modelan en Selenium como un objeto *WebElement*. La búsqueda de elementos se realiza a través del método *findElement* del objeto driver (que recordemos es el objeto que modela el navegador de pruebas).

EjemploSeleniumTest

```
WebElement                                          nombre                                          =
driver.findElement(By.xpath("//span[contains(text(),'Cliente')]"));
```

En este ejemplo se realiza una búsqueda de un elemento en base a una expresión de *XPath*. La clase *By* permite realizar búsquedas en base a los siguientes tipos:

org.openqa.selenium.By

```
/**
 * @param id The value of the "id" attribute to search for
 * @return a By which locates elements by the value of the "id" attribute.
 */
public static By id(final String id);
```

```
/**
 * @param linkText The exact text to match against
 * @return a By which locates A elements by the exact text it displays
 */
public static By linkText(final String linkText);
```

```
/**
 * @param linkText The text to match against
 * @return a By which locates A elements that contain the given link text
 */
public static By partialLinkText(final String linkText);
```

```
/**
 * @param name The value of the "name" attribute to search for
 * @return a By which locates elements by the value of the "name" attribute.
 */
public static By name(final String name);
```

```
/**
 * @param name The element's tagName
 * @return a By which locates elements by their tag name
 */
public static By tagName(final String name);
```

```
/**
 * @param xpathExpression The xpath to use
 * @return a By which locates elements via XPath
 */
public static By xpath(final String xpathExpression);
```

```
/**
 * Finds elements based on the value of the "class" attribute. If an element
 * has many classes then
 * this will match against each of them. For example if the value is
 * "one two onone", then the
 * following "className"s will match: "one" and "two"
 *
 * @param className The value of the "class" attribute to search for
 * @return a By which locates elements by the value of the "class" attribute.
 */
public static By className(final String className);
```

```
/**
 * Finds elements via the driver's underlying W3 Selector engine.
 * If the browser does not
 * implement the Selector API, a best effort is made to emulate the API.
 * In this case, we strive
 * for at least CSS2 support, but offer no guarantees.
 */
  public static By cssSelector(final String selector);
```

Como los Ids de JSF son compuestos en base a su jerarquia de components y las expresiones de XPath para localizarlos pueden ser complejas, se ha creado un método de utilidad para ayudar a localizarlos por Id.

AbstractSeleniumTest

```
/**
 * Obtiene una referencia al elemento cuyo Id de JSF termina en el id
 * parcial pasado.
 * @param partialId Id del elemento JSF sin la parte correspondiente al
 * nombre del formulario
 * y otros elementos que participen en la composición del Id.
 * @return referencia al elemento si este se encontró
 */
protected WebElement getJsfElement(String partialId);

/**
 * Obtiene una referencia al elemento cuyo Id de JSF termina en el id
 * parcial pasado.
 * @param type tipo de elemento Html sobre el que se restringirá la búsqueda.
 * @param partialId Id del elemento JSF sin la parte correspondiente al
 * nombre del formulario
 * y otros elementos que participen en la composición del Id.
 * @return referencia al elemento si este se encontró
 */
protected WebElement getJsfElement(String type, String partialId);
```

Además de los métodos *findElement* de la clase *driver* y *getJsfElement* de *AbstractSeleniumTest*, existe otro conjunto de métodos *findElements* y *getJsfElements* que devuelven una lista de *WebElement* acorde a los criterios de búsqueda (por ejemplo, las filas de una tabla).

5.3.1.3. Insertar texto en un campo

Para insertar un texto en un campo de texto primero se localiza el elemento y luego se utiliza el método *sendKeys*. Se puede hacer todo en una misma línea de código.

EjemploSeleniumTest

```
driver.findElement(By.id("j_username")).sendKeys(user);

getJsfElement("input", "inputTextClienteEstadoCivil").sendKeys("Soltero");
```

Este método añade el texto pasado al valor del campo. Si no se quiere esto, primero se tendrá que vaciar el contenido del campo y después añadir el texto.

5.3.1.4. Leer y vaciar el contenido de un campo

Para leer el contenido de un campo se utiliza el atributo '*value*' del elemento.

Leer contenido de campo

```
// Leer el contenido del 'input' con Id de JSF 'inputTextClienteApellido1'
WebElement texto = getJsfElement("input", "inputTextClienteApellido1");
String valor = texto.getAttribute("value");
```

Para vaciar el campo se utiliza el método *clear()*.

Eliminar contenido de campo

```
// Eliminar contenido de campo 'input'
getJsfElement("input", "inputTextClienteApellido1").clear();
```

5.3.1.5. Hacer click en un enlace o botón

En este caso se usará el método *click()* de la clase *WebElement*.

Ejecutar acción

```
// Ejecutar login
driver.findElement(By.id("enterButtonLink")).click();

// Ejecutar filtrado
getJsfElement("a", "filtrarLink").click();
```

Si el *click* desencadena una carga de página, se navegará hacia esta y se esperará el *timeout* definido para la disponibilidad de esta.

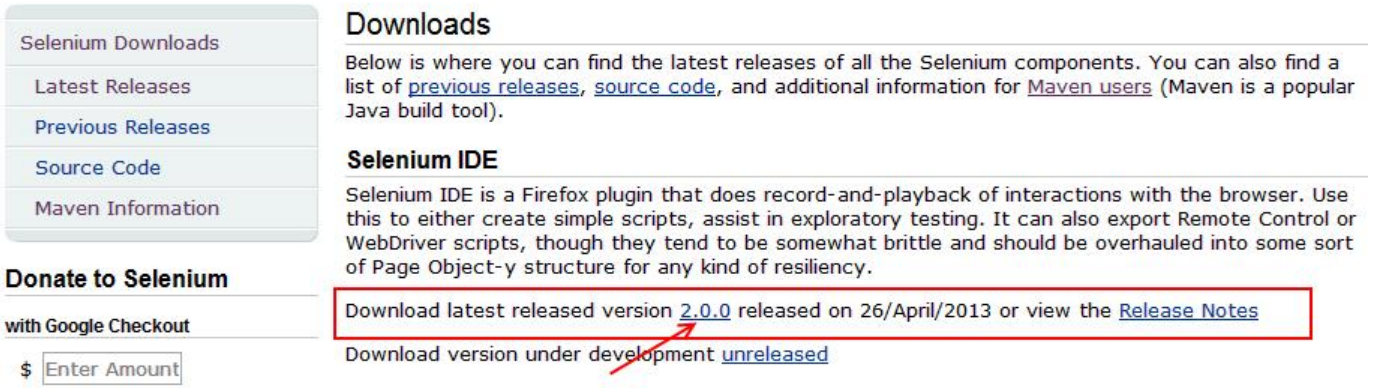
5.4. Creación de un test unitario con SeleniumIDE

SeleniumIDE es un plugin para el navegador *Mozilla Firefox* que **permite grabar la navegación de un usuario en la aplicación web a probar y luego convertir esa navegación en un script java ejecutable**.

Para poder empezar a trabajar con este plugin es necesario instalarlo en Firefox. Debe iniciarse este navegador y acceder a la página de descargas de Selenium:

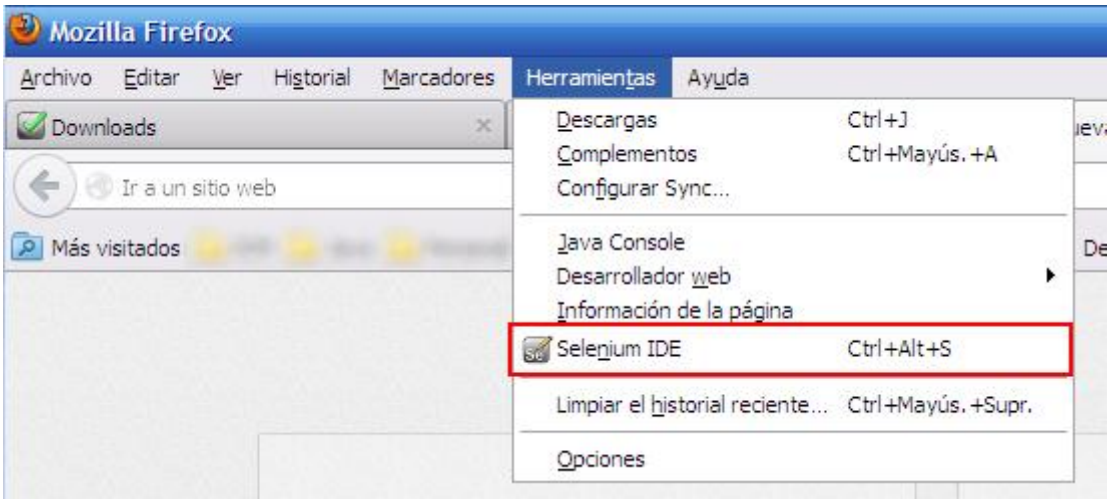
http://docs.seleniumhq.org/download/

A continuación se muestra el enlace de descarga en la página de Selenium para instalar el plugin:

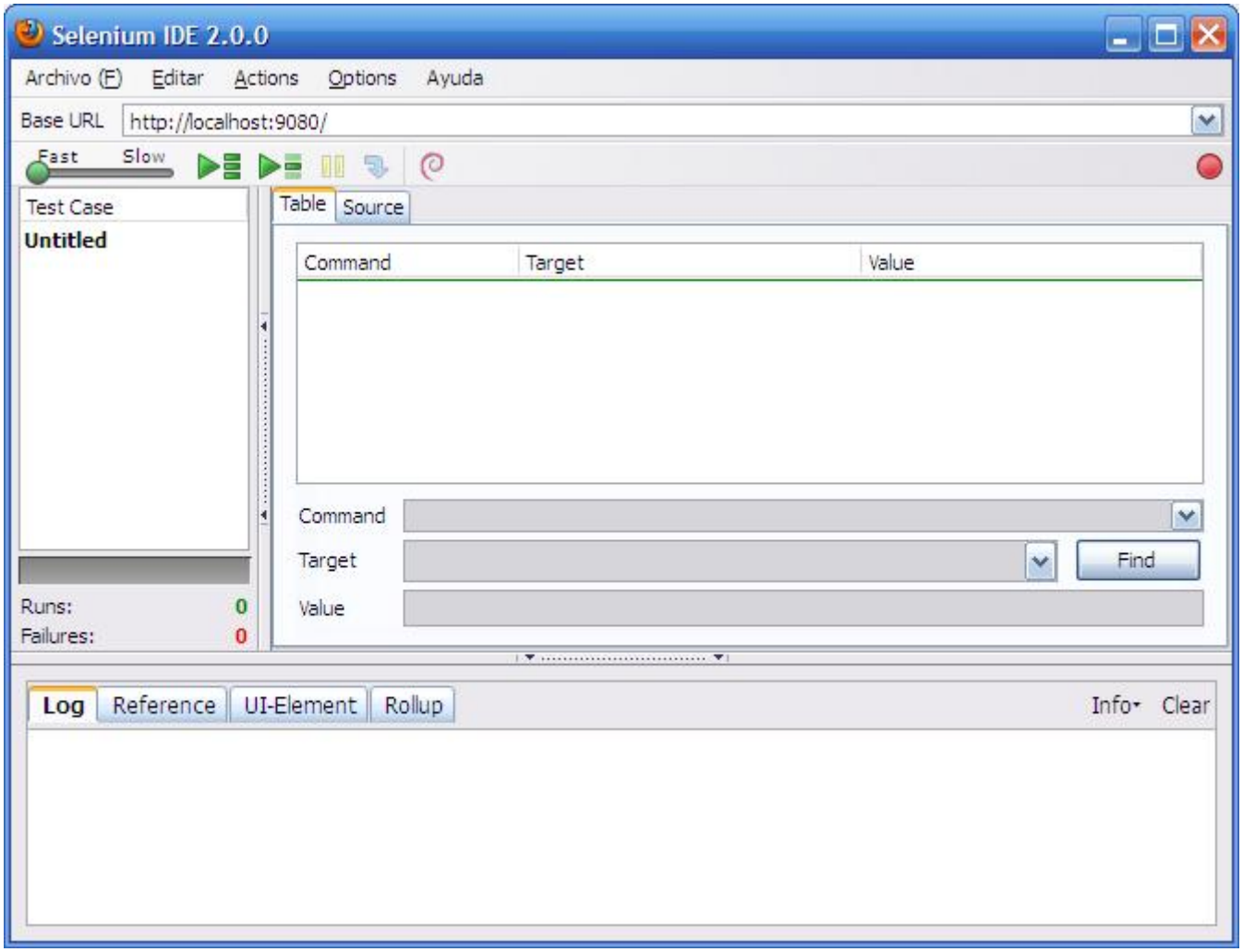

Selenium Downloads
Latest Releases
Previous Releases
Source Code
Maven Information
Donate to Selenium
with Google Checkout
$ Enter Amount
Downloads
Below is where you can find the latest releases of all the Selenium components. You can also find a list of previous releases, source code, and additional information for Maven users (Maven is a popular Java build tool).
Selenium IDE
Selenium IDE is a Firefox plugin that does record-and-playback of interactions with the browser. Use this to either create simple scripts, assist in exploratory testing. It can also export Remote Control or WebDriver scripts, though they tend to be somewhat brittle and should be overhauled into some sort of Page Object-y structure for any kind of resiliency.
Download latest released version 2.0.0 released on 26/April/2013 or view the Release Notes
Download version under development unreleased

Será necesario indicar a Firefox que se permite realizar la instalación del plugin. Se intentarán instalar varios plugins de Selenium IDE; aunque no todos son necesarios, por facilidad se recomienta instalarlos. La página de extensiones del navegador deberá reflejar la instalación de los plugins una vez reiniciado:

Para iniciar el plugin, se seleccionará la opción en el menú Herramientas:



El plugin se muestra en una ventana flotante nueva:



Para iniciar la grabación de un test, basta con pulsar el botón rojo en la parte derecha de la pantalla. Cuando no se está ejecutando una grabación se mostrará el botón y cuando se pulse el botón cambiará a y la grabación comenzará. Al mostrar la pantalla del plugin la grabación estará activada por defecto.

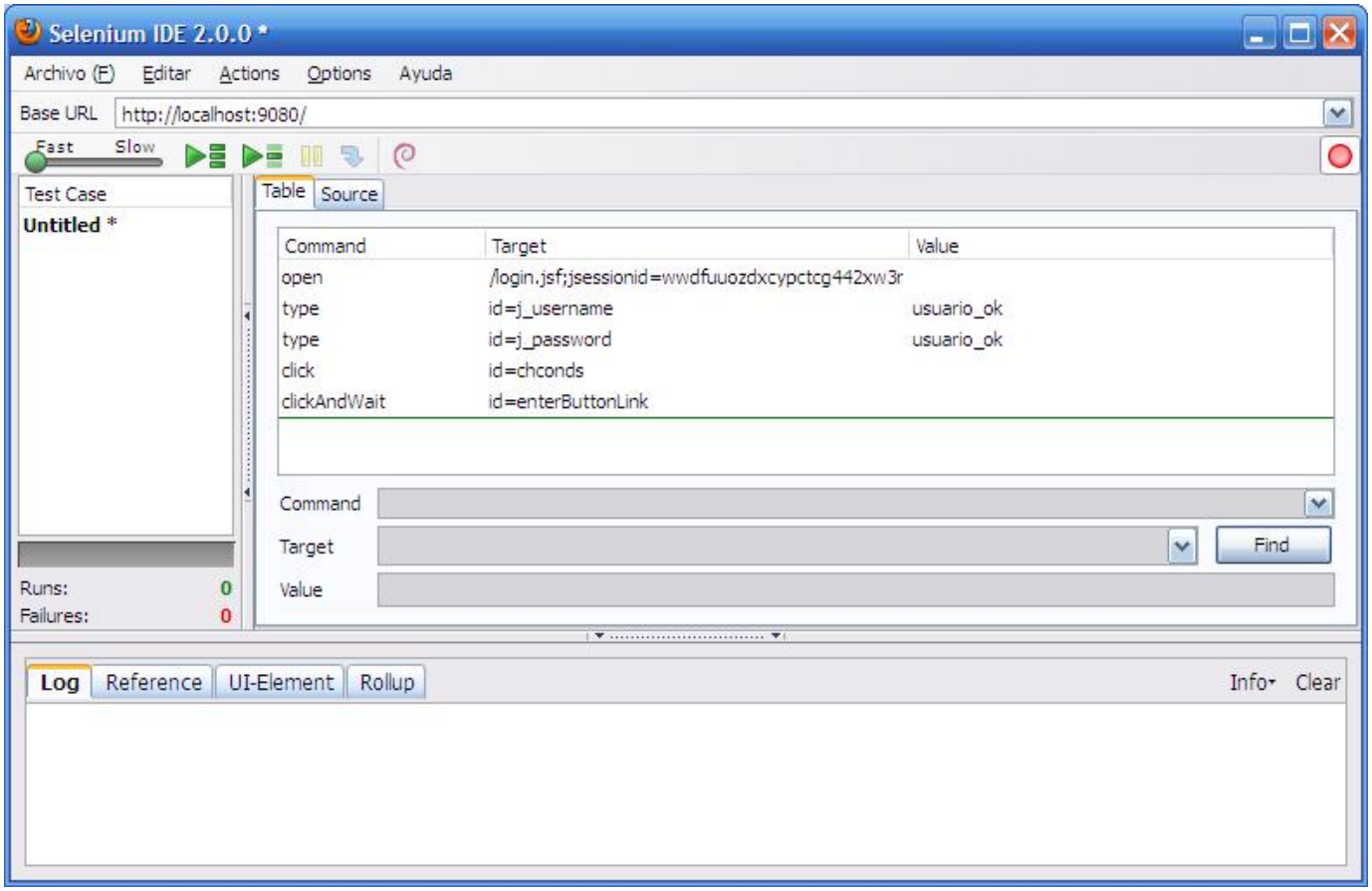
Para iniciar un nuevo test, simplemente ha de pulsarse en la opción *Archivo → New Test Case*.

5.4.1. Ejemplo de test

A continuación se mostrará un ejemplo de cómo crear un test con el plugin SeleniumIDE. Para realizar las pruebas en el entorno local es necesario iniciar un servidor con la aplicación. Recordemos que eso puede conseguirse de forma sencilla con el comando *mvn jetty:run*.

Iniciamos un navegador Firefox y mostramos el plugin de Selenium. El test estará grabando, por lo que no es necesario en este caso pulsar el botón rojo.

Introducimos la URL de la aplicación y realizamos el login. Una vez en la página principal de la aplicación, el aspecto del plugin será el siguiente:



Para verificar que estamos en la página principal pulsamos con el botón derecho sobre 'Home' y seleccionamos la opción '*verifyTextPresent Home'*:

Ahora navegaremos a través de la opción '*Información de usuario'*. Para comprobar en el test que la página visualizada es la correcta, pulsaremos con el botón derecho sobre 'infoUsuario' y volveremos a seleccionar '*verifyTextPresent infoUsuario'*.

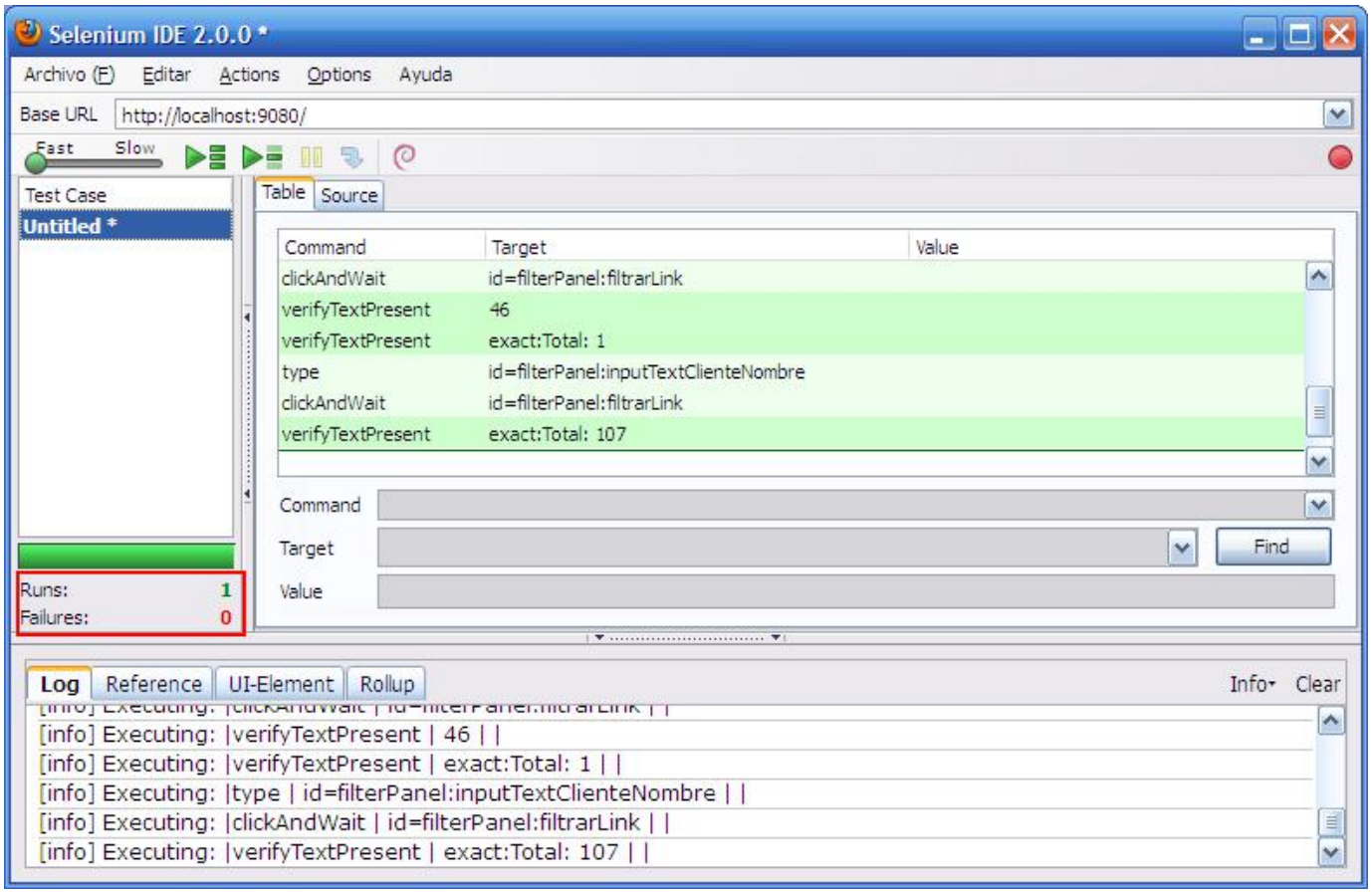
Ahora iremos a Mantenimiento de *Catalogos → Cliente*. Introducimos la palabra "Carlos' en el campo 'Nombre' y pulsamos el botón filtrar. Se mostrará un único registro en la tabla. Pulsaremos con el botón derecho en el texto 'Total: 1' en la parte inferior derecha de la tabla y añadiremos un comando 'verifyTextPresent'. Eliminamos el texto 'Carlos' del campo 'Nombre' y volvemos a pulsar en 'Filtrar'. Repetimos la operación con el texto de total de elementos de la tabla.

Pulsamos el botón de grabación del plugin y con esto damos por concluido el test. A continuación veremos como ejecutar el test, como exportarlo a una clase java y las modificaciones necesarias para parametrizar los datos de entorno.

5.4.2. Ejecución de un test grabado

Una vez grabado el test, comprobaremos que es correcto pulsando el botón . Este botón ejecutará el test que

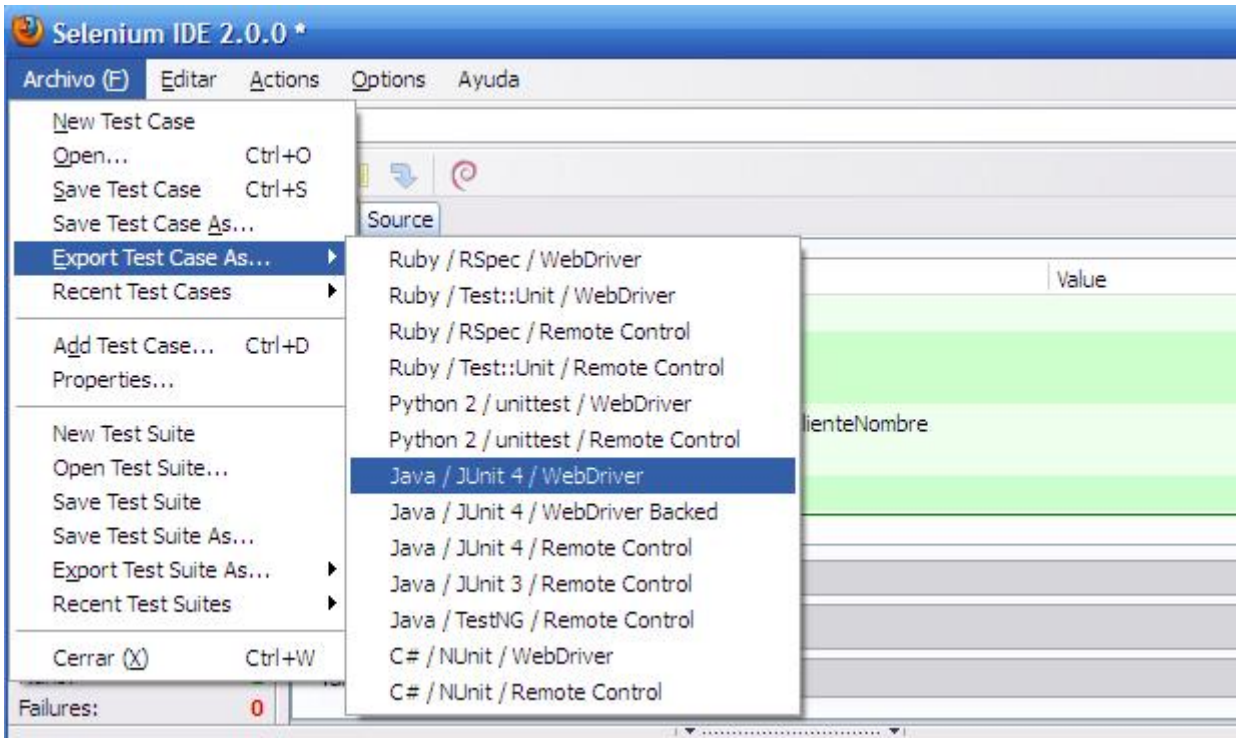
tengamos seleccionado. Si queremos ejecutar todos los tests cargados, pulsaremos el botón en su lugar.



En este caso, vemos que se ha ejecutado un test con éxito.

5.4.3. Exportación del test a una clase java

Para exportar el test grabado a una clase java, lo seleccionamos y en el menú *Archivo* ejecutamos '*Export Test Case as → Java / Junit 4 / WebDriver*'.



Grabaremos el test dentro del proyecto en el paquete de los tests de Selenium (paquete dentro del directorio *src/test/java* del sub-modulo *test* del proyecto).

5.4.4. Adaptar test grabado

Es necesario realizar algunas modificaciones al test para adaptarlo al entorno de tests y poder parametrizar los datos de prueba.

5.4.4.1. Adaptar el paquete de la clase

El test se graba con el paquete '*com.example.tests'*. Adaptarlo a la ruta real del test.

5.4.4.2. Extender la clase AbstractSeleniumTest

El test creado deberá extender la clase *AbstractSeleniumTest* e incluir la anotación *@ContextConfiguration*.

```
Extender AbstractSeleniumText
```

ANTES:

```
public class SeleniumIdeTest {
```

DESPUÉS:

```
@ContextConfiguration(locations = {
        "classpath:/conf/applicationContext-selenium.xml"})
public class SeleniumIdeTest extends AbstractSeleniumTest {
```

5.4.4.3. Eliminar los métodos setUp y tearDown

Eliminar también las variables *driver* y *baseUrl*.

5.4.4.4. Añadir el método getPage

El método *getPage* tiene que ser añadido por exigencia de la clase *AbstractSeleniumTest*. La página de apertura deberá ser parametrizada en el fichero environment.properties. En el ejemplo, la página inicial del test será la página principal de la aplicación.

añadir método getPage()

```
/** Pagina de ejecución del test */
@AutowiredProperty(name="ideTest.page")
private String page;

/**
 * Paso de la página de ejecución del test a la clase padre
 * @return página de test
 */
@Override
protected String getPage() {
    return this.page;
}
```

environment.properties

```
# SeleniumIdeTest
ideTest.page=/secure/index.jsf?id=1
```

5.4.4.5. Sustituir el código de autenticación

El código de autenticación generado por el plugin deberá cambiarse por una llamada al método *authenticate*.

ANTES

```
public void testSeleniumIde() throws Exception {
    driver.get(baseUrl + "/login.jsf;jsessionid=wwdfuuozdxcypctcg442xw3r");
    driver.findElement(By.id("j_username")).clear();
    driver.findElement(By.id("j_username")).sendKeys("usuario_ok");
    driver.findElement(By.id("j_password")).clear();
    driver.findElement(By.id("j_password")).sendKeys("usuario_ok");
    driver.findElement(By.id("chconds")).click();
    driver.findElement(By.id("enterButtonLink")).click();
```

DESPUES

```
/** Usuario para el login. Obtenido de environment.properties */
@AutowiredProperty
private String user;

/** Contraseña para el login. Obtenido de environment.properties */
@AutowiredProperty
private String pass;

@Test
public void testSeleniumIde() throws Exception {
    // Realizar Autenticacion
    authenticate(this.user, this.pass);
```

5.4.4.6. Ejecutar tests java

Solo queda ejecutar los tests de Selenium creados y adaptados para confirmar que todo es correcto, según se indica en el apartado “Ejecución de las Pruebas” de este documento.

5.1. Test unitarios para aplicaciones con certificados

Para poder hacer test unitarios con Selenium de aplicaciones con certificado hay que hacer los siguientes pasos:

5.1.1. Crear un profile de Firefox para las pruebas

Crear un acceso directo a Firexfox con la opción -p

Ejecutar Firefox y en la pantalla de selección de perfil pulsar en “crear perfil…” y darle un nombre al nuevo perfil

5.1.2. Importar el certificado a usar

Sólo se debe importar el certificado con el que se vayan a hacer las pruebas

En la pantalla de opciones / avanzado / cedrtificados dejar seleccionado el check “Seleccionar uno automáticamente”

Ahora al entrar en la aplicación por https seleccionará automáticamente el certificado que tenemos seleccionado y entrará.

5.1.3. Hacer los test de Selenium con el nuevo perfil por defecto

Para hacer las pruebas seleccionar el perfil que hemos creado y marcar “No preguntar al inicio”

Hacer las pruebas de selenium con esta configuración

Al terminar las pruebas volver a pulsar en el acceso directo que tenía la opción –p para quitar la opción “No preguntar al inicio”

6. ENLACES RELACIONADOS

Producto	URL
Selenium	http://docs.seleniumhq.org/